

Loading vc-cvs...

```

1 package caltool.schedule;
2
3 import caltool.PrecondViolation;
4 import java.util.*;
5
6 /**
7  *
8  * Class ScheduleEventPrecondViolation defines an exception containing error
9  * conditions for the Schedule.scheduleEvent method. It contains a list of
10 * the specific error messages that may be output in response to a precondition
11 * having been violated by a call to ScheduleEvent.
12 *
13 */
14 public class ScheduleEventPrecondViolation extends Exception
15     implements PrecondViolation {
16
17     /**
18      * Construct this by initializing the error message list to an empty list,
19      * initializing the numErrors count to 0, and initializing local copies of
20      * the error message text for each of the possible errors from
21      * Schedule.scheduleEvent.
22      */
23     public ScheduleEventPrecondViolation() {
24
25         errors = new ArrayList();
26
27         emptyTitleMessage = new String(
28             "Event title cannot be empty.");
29         alreadyScheduledMessage = new String(
30             "An event of the given start date and title is already scheduled.");
31         invalidStartDateMessage = new String(
32             "Invalid start date.");
33         invalidEndDateMessage = new String(
34             "Invalid end date.");
35         noActiveCalendarMessage = new String(
36             "There is no active calendar in the Calendar Tool workspace.");
37
38         numErrors = 0;
39     }
40
41     /*-
42      * Implemented interface methods.
43      */
44
45     /**
46      * Return the error list.
47      */
48     public String[] getErrors() {
49         return (String[]) errors.toArray(new String[1]);
50     }
51
52     /**
53      * Clear all error messages.
54      */
55     public void clear() {
56
57         errors = new ArrayList();
58         numErrors = 0;
59     }
60
61     /**
62      * Return true if any errors have been set.
63      */
64     public boolean anyErrors() {
65         return (numErrors > 0);
66     }
67
68     /**
69      * Return the number of errors.
70      */
71     public int numberOfErrors() {
72         return numErrors;
73     }
74
75     /*-
76      * Error-setting methods
77      */
78
79     /**
80      * Set the empty title error message.
81      */
82     public void setEmptyTitleError() {
83         errors.add(emptyTitleMessage);
84         numErrors++;
85     }
86
87     /**
88      * Set the already scheduled error message.
89      */
90     public void setAlreadyScheduledError() {
91         errors.add(alreadyScheduledMessage);
92         numErrors++;
93     }
94
95     /**
96      * Set the invalid start date error message.
97      */
98     public void setInvalidStartDateError() {
99         errors.add(invalidStartDateMessage);
100        numErrors++;
101    }
102
103    /**
104     * Set the invalid end date error message.
105     */
106    public void setInvalidEndDateError() {
107        errors.add(invalidEndDateMessage);
108        numErrors++;
109    }
110
111    /**

```

```
112     * Set the no active calendar error message.
113     */
114     public void setNoActiveCalendarError() {
115         errors.add(noActiveCalendarMessage);
116         numErrors++;
117     }
118
119
120     /*_*
121     * Data fields
122     */
123
124     /** List of current error messages */
125     protected ArrayList errors;
126
127     /** Error message count */
128     protected int numErrors;
129
130
131     /** Error message for empty title */
132     protected String emptyTitleMessage;
133
134     /** Error message for event of same date,title already scheduled */
135     protected String alreadyScheduledMessage;
136
137     /** Error message for invalid start date */
138     protected String invalidStartDateMessage;
139
140     /** Error message for invalid end date */
141     protected String invalidEndDateMessage;
142
143     /** Error message for no currently active calendar in the workspace */
144     protected String noActiveCalendarMessage;
145
146 }
```